Characterizing Dynamic Neural Networks - A study on the effect of batch size on model training metrics for RNNs

Ashish Hooda, Nithin Venkatesh, Sangeetha Grama Srinivasan - Group 8

December 2019

1 Introduction

Deep neural networks (DNN) have gained widespread attention because of the vast and diverse applications that they cater to. DNNs today power most of the solutions behind both business and research problems including online search, content recommendation, natural language processing and computer vision problems. A subset of DNNs called the Dynamic neural networks whose architecture can change with the input are becoming increasingly popular owing to their use in systems for machine translation and modelling sequential data. Although much work has been done on characterizing the resource footprints with varying mechanisms for DNNs, little has been accomplished about resource characterization of recurrent neural networks. In this work, we study resource utilization characteristics of dynamic neural networks like RNNs. We aim to tune the hyperparameters such as batch size and learning rate based on the GPU utilization metrics which can be obtained with a single epoch of training.

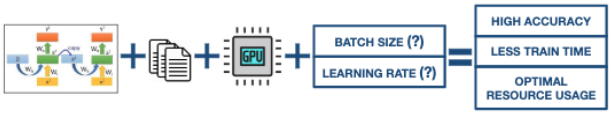

Figure 1: Overview of our study

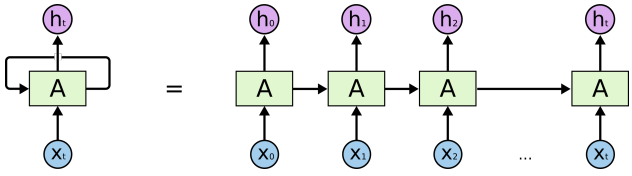

Figure 2: RNN architecture

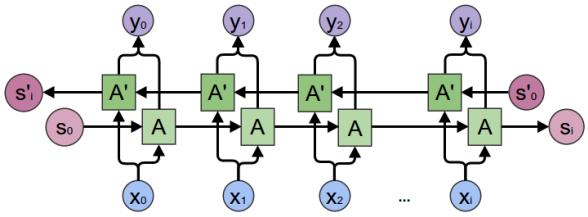

Figure 3: Bi-Directional RNN architecture

2 Background

Recurrent Neural Networks, are a type of Deep Neural Networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. [1] In other words, RNNs retain contextual information while processing new data. This temporal behaviour in RNNs allows information to be passed from one time step to the next. The unrolled version of RNNs is as shown in the Figure-1, where the RNNs can be thought of as a collection of replicas of the same model, each passing information to the next. To overcome the problems associated with RNNs such as short-term memory retention and vanishing gradients [1] [2], Long-Short term memory (LSTMs) [3] and Gate Recurrent Units (GRUs) [4] have been proposed. They work similar to RNN cell with additional gates that selectively allow information to be passed on from one step to the other. Despite simple architecture of RNNs as compared to other networks like Convolutional neural networks (CNNs) [5], RNNs have a lot more model parameters, and training these models is time consuming and memory intensive.

LSTMs have been extensively used in production environments for various machine learning tasks like natural language translation, time series analysis, image and video captioning, speech to text and text to speech workloads. The training time for these models are large owing to the large model parameter size and the large amount of data that these models deal with. Both the large input data as well as the large number of parameters make these models highly memory intensive. In this report, we

aim to study and characterize the memory and compute footprint of these models, on the training task of natural language translation [6]. We concentrate on English to French language translation [7] data set for the characterization.

We use two types of model architectures and train them on two GPU architectures (Nvidia K80 and Nvidia P100). The models we train are Bi-directional RNNs, the architecture for which is as shown in figure 3 and Encoder-Decoder model for Machine Translation. Bi-directional RNNs can be understood as a combination of 2 independent RNNs where the input is supplied in reverse order for one of the networks. The outputs of the two RNNs are combined using aggregation techniques like summation or concatenation. Encoder-Decoder RNN architecture consists of an encoder, which encodes the input sequence into a fixed-length representation, and a decoder, which decodes the representation into an output sequence. Note that from this point forward, we will refer to RNNs/LSTMs/GRUs as RNNs in general.

We take a two step approach in the project. First, we characterize the metrics of validation accuracy and training time for RNN training on a subset of English to French translation. Hyper-parameters like batch size and learning rate play a significant role on model performance affecting metrics like accuracy and time taken to train (Which is associate with the cost the user has to pay for resources). Most engineers choose hyper-parameter values either on trial and error basis or based on previous literature. This approach may lead to large search space of hyper parameters decreasing developer productivity. In addition to characterizing RNN training, we aim to study a more efficient way of hyper parameter search. Using GPUs available in CloudLab [19] and Google Compute Engine [11], we train different RNN models on different GPUs and study their behaviour. We mainly concentrate on how hyper parameters like batch size and learning rate affect model metrics, such as final validation accuracy and training time. We measure GPU utilisation and observe how the batch size affects GPU utilization and how the GPU utilization information can be used to come up with the optimal hyper parameters. Based on the characterization made, we propose a greedy technique for batch size and learning rate search and evaluate the same as shown in Figure-4.

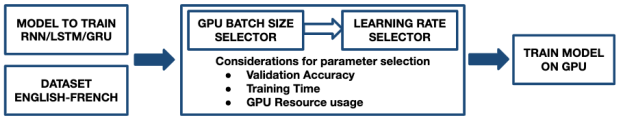

Figure 4: Design of our tool to select Batch Size and Learning Rate

3 Design

Batch Gradient Descent provides a smoother convergence towards the optimal value while Stochastic Gradient Descent has the benefit of converging faster. Moreover, for large datasets that don't fit in memory, memory latency is a bottleneck whereas computation is abundant which favours batching of gradient calculation operations. This calls for an optimal batch size which would ensure a smooth convergence as well as a higher value for operations per second.

To select an efficient batch size for a given hardware, we claim that the an optimal selection can be made based on the hardware utilisation. We concentrate on GPUs for this work. This claim is based on the fact that higher GPU utilization will result in more operations per second reducing the time taken per training sample. This means that the model can be trained for a higher number of epochs in the same time, increasing accuracy. However, a larger batch size may lead to slower convergence as the individual gradients cancel out to give a smaller gradient value for the batch. We can compensate this reduction in gradient by increasing the learning rate as proposed by Goyal et al. [8].

3.1 System Design

3.1.1 Model architecture

We build two RNN architectures for characterization. The Encoder-Decoder model has two layers of 128 GRU cells. The Bi-Directional RNN has one layer of 128 bidirectional GRU cells. Both these models use sparse categorical crossentropy metric and Adam optimizer to reduce the loss metric. Both the models have final time distributed layer, that is responsible to produce the output word probability vectors for a given input word. The total number of model parameters in case of GRU architectures is dominated by the last layer. Precisely, the number of model parameters in the last layer is (number of GRU cells + 1) * (Target language vocabulary size + 1). For large dataset with vocabulary size in the range of 10K and with 100 GRU cells in the last layer, the number of model parameters can reach to 1M.

3.1.2 GPU architectures used

We set up an instance with Nvidia K80 GPU on Google Compute Engine[11]. We also used an instance from CloudLab [19] with the Nvidia P100 GPU. The Nvidia K80 GPU has 26 Streaming Multiprocessors, and can support around 2.8TFLOPS (Terra-FLOPS). The Nvidia P100 GPU has 56 Streaming Multiprocessors, and supports around 4.7TFLOPS. Both the GPUs that we obtained have 12GB memory. The P100 GPU has a mem-

ory bandwidth of 720GB/s and while the K80 has a memory bandwidth of 480GB/s. We measured the compute and memory utilisation of both the GPUs using the Nvidia-smi command available in the Nvidia-cuda toolkit. Keras uses Tensorflow-GPU package to train models using GPU. Utilisation of GPU memory as measured by Nvidia-smi is defined as *"Percent of time over the past sample period during which global (device) memory was being read or written."*. The GPU compute utilisation can be defined in a similar fashion, as given in [13].

3.2 Dataset used

We selected two English-French datasets. The datasets differ in vocabulary size. The smaller dataset has approx. 10^5 pairs of English-French sentences. We took the dataset from the European Parliament Proceedings Parallel Corpus, 1996-2011 [9]. We pre-process the data using tokenizing and padding. Since the dataset did not fit in memory for both the GPUs, we used thresholding policies to limit vocabulary based on a sentence length and word occurrence frequency. The amount of data to be maintained in memory is linearly proportional to the (target language vocabulary size) * (input sentence length) * (batch size), which grows drastically with the batch size. So we limit the input sentence to 100 and in section 3.3, we show how the target language vocabulary size varies with various techniques.

3.3 Effect of vocabulary size on RNN model training

The complexity of a Machine Translation task depends on the vocabulary sizes of the source and target language. The source vocabulary size determines the input dimension and the target vocabulary size determines the output classes. Moreover, the number of parameters is directly proportional to the output vocabulary size. To analyse this dependence, we conduct experiments with two different datasets having different vocabulary sizes. For the larger dataset, we perform vocabulary pruning using two techniques - Byte Pair Encoding and Common Word Elimination. The effect of common word elimination with increasing in threshold on word count is as per figure 5

Byte Pair Encoding starts from a vocabulary comprising of the ASCII characters and iteratively increasing the vocabulary by merging the most commonly occurring token pairs. This process reduces the skewness of frequency counts of different tokens in the vocabulary. Also, this helps in tuning the vocabulary to a desired size by limiting the number of merge iterations. On the other hand, common word elimination reduces the vocabulary

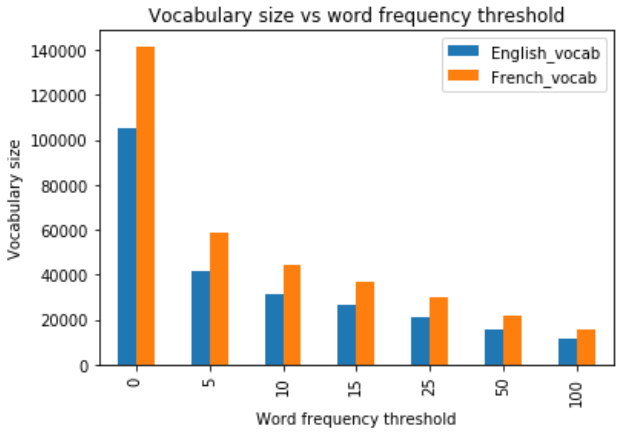

Figure 5: Change in vocabulary size with word frequency threshold

size by substituting the most frequently occurring tokens. The technique preserves the less frequent words as they carry the most contextual information.

For Common Word Elimination, we analysed how the vocabulary size changes with different values of the threshold for most commonly occurring tokens. The variance of vocabulary size with threshold is outlined in [12]

3.4 Hyper parameter probing

We devised a search grid, with batch size ranging from 2^9 to 2^{14} (6 points) and learning rates ranging from 10^{-5} to 10^{-1} (5 points). We developed driver scripts to train both the models on all these search grid points using the Nvidia K80 GPU on the smaller dataset. We repeated this on the Nvidia Tesla P100 GPU and collected metrics and usage information for each of the 30 points on the search grid. We also collected the utilisation metrics using the nvidia-smi tool. We collected both the GPU compute and memory utilisation metrics. The scripts and commands used to run the experiments can be found here. [10]

To verify if the trend is dependent on the dataset used, we used the bigger dataset to train the two models. Owing to the large vocabulary size of the dataset, it needed more memory than what was available on both the GPUs (12GB), we used batch processing where in data is loaded from RAM to GPU memory based on the batch size.

Based on the characterization we observed from the search grid plot as presented in 4, we device a greedy technique to suggest the optimal batch size and learning rate for a given model architecture. The technique is as follows:

- Choose the batch size from the list of possible batch sizes that gives the best GPU utilization. We take this batch size to be first one that has more than 90 percent of the maximum achievable GPU utilization. This can be done by training for one epoch and getting the GPU utilization metrics.
- For the above chosen batch size, vary the learning rate in log scale and choose the learning rate that gives the best saturation accuracy (using early stopping). This trend can also be observed on a sample of the data to reduce the time requirements.
- Once the best learning rate from the log scale is chosen, now try learning rates with linear scaling around the chosen log scale learning rate, i.e if 0.001 is the optimal learning rate in log scale, we linearly scale it as 0.001, 0.002 etc. until we reach the local maximum in validation accuracy around the chosen log scale learning rate.
- We present the evaluation of this greedy technique in section 4

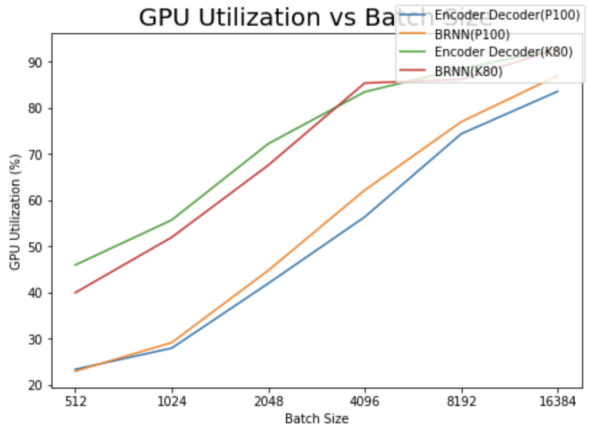

Figure 6: GPU Utilisation trend for model training with varying batch sizes.

4 Evaluation

4.1 Resource Utilisation

Figure-6 shows the GPU utilisation (compute) trend on varying batch size. The red and green lines (upper pair of lines) are plotted based on utilisation of the P100 GPU for both models. The orange and blue (lower pair of lines) show the same for K80 GPU. For both GPUs, we see that irrespective of model architecture, the utilisation increases as batch size is increased. This is mainly because, the number of samples in every batch increases, and hence compute increases with batch size. The utilisation for K80 is higher than P100 since K80 has a lower compute capacity when compared to the P100 GPU. This means, for the same operation, the compute utilisation of K80 will be higher than for P100.

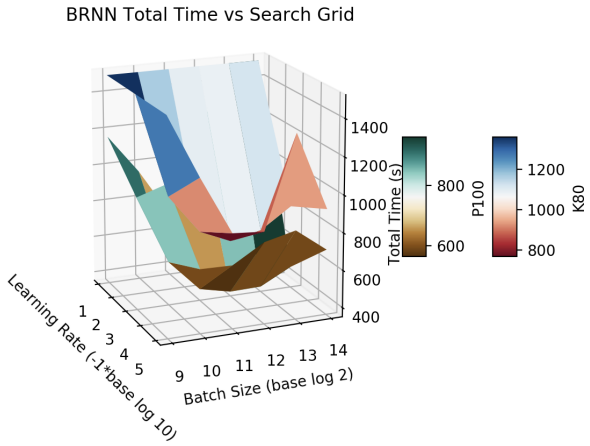

Figure 7: Total training time comparison for both GPUs

The graph given by 7 plots the total time taken to train the Bi-directional RNN model over different search grid points between the P100 and K80 GPUs. The search grid is denoted by the Batch Size and Learning Rate axes. This graph shows that at any given point in the search grid, the P100 GPU outperforms the K80 GPU (owing to the higher compute capacity). The graph also reiterates the fact that since different hardware types have varying compute capacities, a batch size which is good for the P100 might not be optimal for K80.

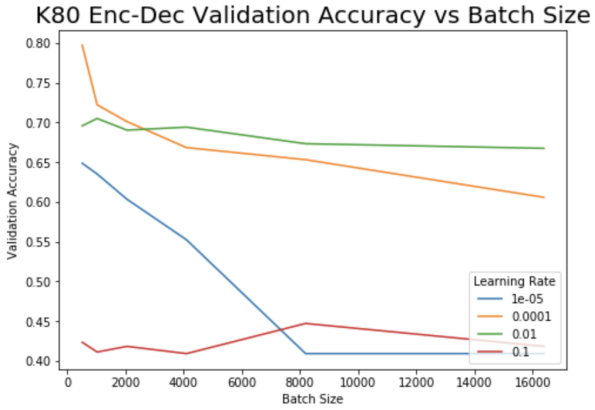

Figure 8: Variation of Model Accuracy for different batch sizes and learning rates on K80 GPU for Encoder Decoder Model

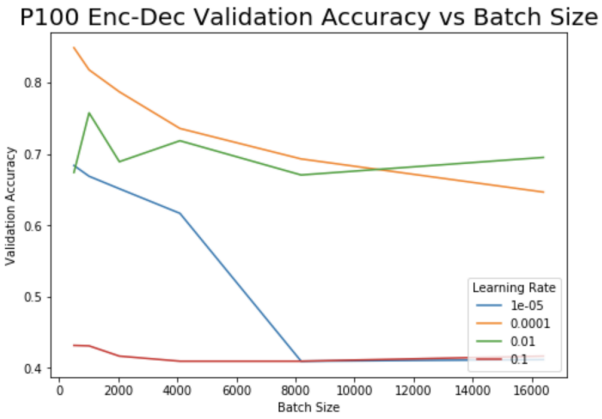

Figure 9: Variation of Model Accuracy for different batch sizes and learning rates on P100 GPU for Encoder Decoder Model

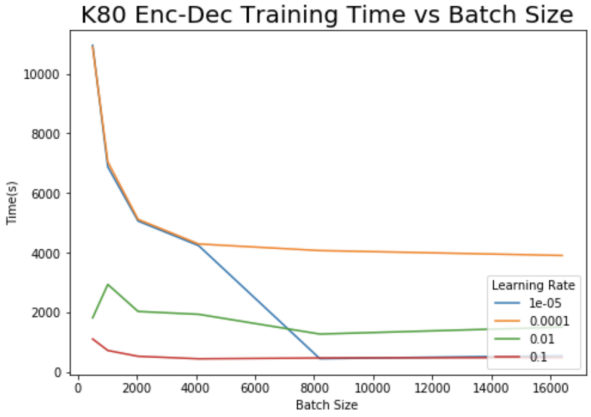

Figure 10: Variation of Model Training Time for different batch sizes and learning rates on K80 GPU for Encoder Decoder Model

4.2 Accuracy

Figure 8 shows the how accuracy varies with different batch sizes for a constant learning rate. Across different learning rates, we see that validation accuracy decreases as batch size increases. We see a similar trend for the P100 GPU. We also see that the highest accuracy achieved for P100 is higher than that of K80. Further, we observe that for smaller learning rate values, validation accuracy decreases drastically with increase in Batch Size. This means larger batch sizes work better with larger learning rates. This can be inferred from the fact that for a large batch size the gradients cancel out to give a smaller resultant gradient value for the batch. To compensate for a smaller gradient, a large learning rate works better.

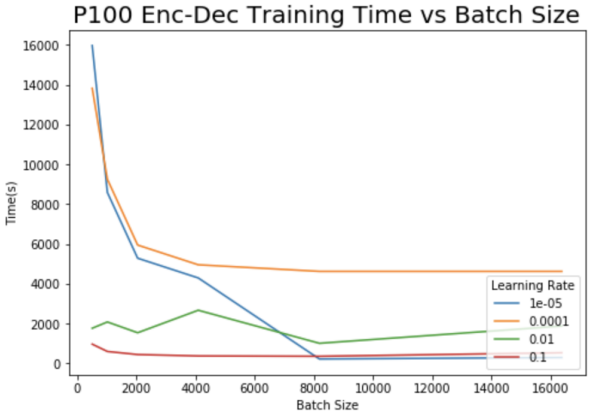

Figure 11: Variation of Model Training Time for different batch sizes and learning rates on P100 GPU for Encoder Decoder Model

4.3 Training Time

Though Training Time depends on the user provided value for number of epochs, we use early stopping since the validation accuracy saturates after a certain number of epochs. Thus, we denote the training time as this time taken for the validation accuracy to saturate. We accomplish this by specifying a patience value to the EarlyStopping functionality in TensorFlow. We collect total training time for various configurations by varying the learning rate, batch sizes accross the search grid.

Figure-10 and Figure-11 show the variation of total training time for different batch sizes and learning rates on the K80 and P100 GPUs respectively. We see that as batch size increases, the total training time initially decreases and saturates after a point. In simpler terms, initially the training time decreases for increase in batch size but as batch size is increased further, the rate of decrease in training time reduces, thus saturating. When the batch size is less, the model is bound by the Von Neumann bottleneck i.e. by the rate at which the next batch can be fetched. As batch size increases, this bottleneck shifts to computation. After some batch size, any further increase in batch size will lead to similar training time since now the model is bound by compute.

4.4 Optimal batch size and learning rate selection

We use the technique proposed in the Design section to choose the batch size and learning rate for Encoder-decoder model with 2 layers of 128 GRU cells on K80 architecture. We observe that at batch size 8192 as in Figure-6, the GPU utilization reaches 90% and we greedily choose this as the optimal batch size for prediction.

We then vary the learning rate on log scale for batch

size 8192 and choose the order 0.001 to be the best range for learning rate based on validation accuracy as shown in Figure- 12.

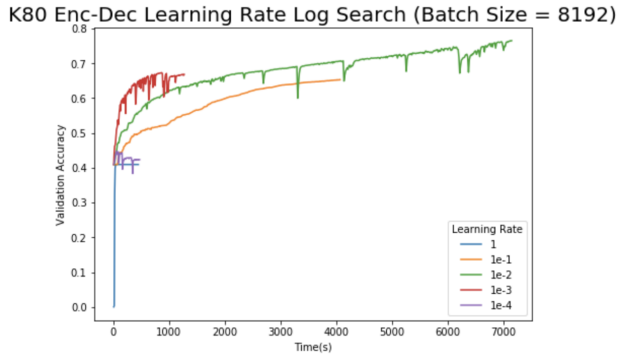

Figure 12: Log scale learning rate accuracy for optimal batch size

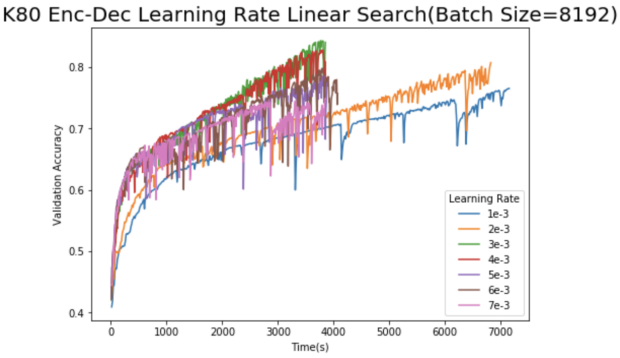

Figure 13: Linear scale accuracy for for optimal batch size and optimal log scale learning rate

We then linearly increase the learning rate and observe that we reach the local minimum at learning rate 0.003 as observed in figure 13. So, the selection procedure gives batch size of 8192 and learning rate 0.003 as the optimal selection.

We observe that this configuration has an accuracy of 84% which is around 6% higher than 79% which is the validation accuracy in the search grid with varying batch sizes and learning rate (log scale). The 79% was obtained at batch size 512 and learning rate 0.0001. We also see that the time for the optimal configuration proposed is 64 minutes as opposed to 180 minutes for the grid configuration, around one-third of the time required to achieve similar accuracy. This can be attributed to the larger batch size of 8192 compared to 512.

5 Related Work

Khomenko et al [15] studied the effect of batching on training time conducting multiple experiments by varying batch sizes for a variety of models. They show that the training time has an inverse relationship with the batch size until an optimal value for the batch size. In other words, the training steps required to achieve an error less than a threshold varies linearly with the batch size (which can be referred to as perfect scaling) and it might saturate after a point (maximum value for data parallelism). However, this optimal value varies for different settings and depends on multiple factors such as model architecture, training framework and data workload. Studying this in context to RNNs is an interesting avenue as they have a strong dependence on input length and process inputs very differently when compared to other Neural Networks.

Shallue et al. [20] studies the effect of sequence bucketing on RNNs using multi-GPU data parallelism. The paper buckets the input data by putting inputs of the same length into a bucket and uses data-parallel training on GPU to train the model on these buckets in parallel. The paper uses a dynamic programming approach to split the input into different buckets. However, the paper does not consider different ordering of buckets in its training experiments. Since RNNs have a strong dependence on the structure of input, we aim to study the effect of batching input samples and the batch sizes based on either length or any other parameter on RNN model performance.

As presented in [18], batch size plays an important role in deciding the convergence time for any Machine Learning training model. Increasing batch size reduces the time to converge as better GPU or CPU utilization compensates for the increase in number of epochs required to converge. However, this plateaus after an optimal batch size suggesting a limit to which the data parallel approach can be exploited. This optimal batch size depends not only on the underlying workload but also the model architecture as well as the training framework. This is even more interesting for the case of RNNs [17] as the input data has varying length which calls for more involved batching to improve computation efficiency. Frameworks like Tensorflow are best suited for static data flow graphs, and once the graph is defined, different batching strategies can be tried to obtain the optimal batch size. For the case of dynamic neural networks, other frameworks like DyNet [14] have attained popularity owing to their features like dynamic graph construction and dynamic batching. We would like to study the overhead of dynamic graph construction on frameworks like Tensorflow and compare with DyNet in terms of resource footprints.

HyperBand[16] proposes a pure exploration non-

stochastic approach to hyper parameter optimization. It covers optimization over a large range of hyperparameters and provides theoretical guarantees. However, we aim at exploiting the GPU utilization metrics to limit the search space and build an easy to configure tool for the same.

6 Conclusion and Future work

In this project we study the effect of batch size and learning rate on model accuracy and training time. We show through our experiments that hyperparameter selection can have a significant impact on model accuracy and training time. Instead of guessing the values for these parameters or using values from previous work, we propose a simple and efficient way to select batch size based on the hardware utilisation and tune the learning rate for the selected batch. Our experiments show that the results collected support our hypothesis.

From the data collected, we observed the following trends:

- Increasing the batch size leads to an increase in GPU compute and memory utilization
- Increasing the batch size leads to a decrease in training time
- Batch size is inversely related to the model accuracy.
- Large values of learning rate work better with large batch sizes.

The study can be extended to include multi-GPU resource utilisation and test the validity of our hypothesis in a multi-GPU set up. It will also be interesting to test our hypothesis in a heterogeneous set-up comprising of CPU and GPUs for training. We plan to study how the validation accuracy trends vary across the larger data set and a sampled data set.

7 Acknowledgment

We want to thank CloudLab and Google Compute Engine for helping us run our experiments using their resources.

References

[1] https://en.wikipedia.org/wiki/Recurrent_neural_network.

[2] https://en.wikipedia.org/wiki/Vanishing_gradient_problem.

[3] https://www.bioinf.jku.at/publications/older/2604.pdf.

[4] https://arxiv.org/pdf/1412.3555.pdf.

[5] https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.

[6] https://en.wikipedia.org/wiki/Machine_translation.

[7] http://www.statmt.org/europarl/.

[8] https://arxiv.org/abs/1706.02677.

[9] European parliament proceedings parallel corpus 1996-2011. http://www.statmt.org/europarl/.

[10] Git repository for the project. https://github.com/sanggs/Characterising-resource-usage-for-RNN/.

[11] Google compute engine. https://cloud.google.com/compute/.

[12] How to prepare a french-to-english dataset for machine translation. https://machinelearningmastery.com/prepare-french-english-dataset-machine-translation/.

[13] Nvidia utilisation struct reference. https://docs.nvidia.com/deploy/nvml-api/structnvmlUtilization__t.html#structnvmlUtilization__t.

[14] Dynet: The dynamic neural network toolkit. 2017.

[15] V. Khomenko, O. Shyshkov, O. Radyvonenko, and K. Bokhan. Accelerating recurrent neural network training using sequence bucketing and multi-gpu data parallelization. 2016.

[16] G. D. A. R. Lisha Li, Kevin G. Jamieson and A. Talwalkar. Efficient hyperparameter optimization and infinitely many armed bandits. 2016.

[17] G. Neubig, Y. Goldberg, and C. Dyer. On-the-fly operation batching in dynamic computation graphs. *ArXiv*, abs/1705.07860, 2017.

[18] I. Pettum. Intro to dynamic neural networks and dynet.

[19] R. Ricci, E. Eide, and The CloudLab Team. Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications. *USENIX ;login:*, 39(6), Dec. 2014.

[20] C. J. Shallue, J. Lee, J. Antognini, J. Sohl-Dickstein, R. F. frostig, and G. E. Dahl. Measuring the effects of data parallelism on neural network training. 2019.